A SYSTEM AND METHOD FOR GENERATING SOURCE CODE FROM A GRAPHICAL MODEL

Field of the Invention

The illustrative embodiment of the present invention relates generally to the use of custom storage classes in a graphical model, and more specifically to the use of a graphical user interface to select parameters for custom storage classes used in the automatic generation of source code from a graphical model and the display of that generated code.

Related Applications

The illustrative embodiment of the present invention is related to a United States Patent Application entitled *"Generating Code For Data References"*, United States Patent No. 09/876, 487, the contents of which are hereby incorporated by reference.

Background

Automatic code generation is a process whereby software source code is automatically produced from a graphical model. The software source code produced by the automatic code generation process may be compiled and then executed on a digital computer or other electronic device implementing the functionality specified by the model. The graphical model being studied may contain many different types of data references. Data may be used to represent the states of the system, as well as the flow of matter, energy, or information between system components. Each item of data in the model is defined to have a data storage class. Data is in turn represented in the generated software source code in a manner that is prescribed by its storage class. The software source code references data in a number of different ways including defining data, declaring data, initializing data, reading a value of data,

assigning the value of data, and the choice of storage class controls how each of these references are generated..

Code generators may provide predefined sets of storage classes, and they may also permit the user to define new, custom storage classes with user-defined characteristics. The custom storage class provides a central point for software code generation of a set of objects since changes to the unique set of instructions defining a custom storage class collectively apply to the (potentially large) set of data of that class. Common software engineering practices that may be enabled with custom storage classes include embedding a data item in a bit field, embedding a data item in a structure, embedding a data item in a union, using platform-specific declarations in the data declaration, defining the scope and storage of the data, declaring data using arbitrary C types, and accessing data through function calls.

Unfortunately, it is often difficult to create a custom storage class. Conventional methods of creating custom storage classes for use with an automatic code generator require the use of programmatic callbacks that interface to the lower levels of the code generator architecture. The writing of the programmatic callbacks requires an in-depth knowledge of the workings of the automatic code generator. The required level of knowledge regarding the automatic code generator is often lacking for many users as is the desire to write the programmatic callbacks.

Brief Summary of the Invention

The illustrative embodiment of the present invention provides a graphical user interface that enables a user to create and specify the properties of custom storage classes. The characteristics of each storage class are specified via parameter settings accessible from the graphical interface. Each custom storage class designed through the interface is defined by a combination of parameter settings. A code preview window in the graphical interface displays the source code references to model data given the selected parameter settings. The display is shown dynamically and adjusted to reflect subsequent changes in parameter settings.

In one embodiment, in an electronic device with a graphical modeling and execution environment that includes at least one graphical model, a method provides a user interface which has different user-selectable parameter settings for a custom storage class. The custom storage class specifies the manner in which an automatic code generator creates source code for data referenced by a graphical model. Following the selection of parameters through the interface, a custom storage class is created using the selected parameters.

In another embodiment, in an electronic device having a modeling and execution environment that includes at least one graphical model, a system includes a user interface with different selectable parameter settings for a custom storage class. The custom storage class specifies the manner in which an automatic code generator creates source code from the model. The system also includes a custom storage class created utilizing parameters selected by a user from the user interface and a view of code generated by the automatic code generator utilizing the user-selected parameters.

Brief Description of the Drawings

Figure 1 depicts an environment suitable for practicing the illustrative embodiment of the present invention;

Figure 2 is a flowchart of the sequence of steps followed by the illustrative embodiment of the present invention to create and display source code referencing model data;

Figure 3 is a flowchart of the sequence of steps followed by the illustrative embodiment of the present invention to adjust and re-display source code generated following a change in the selected parameters for a custom class;

Figure 4 depicts a view of the user interface of the illustrative embodiment of the present invention

Detailed Description

The illustrative embodiment of the present invention provides a user interface displaying user-selectable parameter settings to be used in the creation of a custom storage class. The custom storage class is utilized by an automatic code generator in a modeling and execution environment to create source code referencing data from a graphical model. Salient aspects of the generated source code are displayed to the user in a code view area of the user interface. Subsequent alterations in the user-selected parameters cause a regeneration of the displayed source code and an update of the display of this new code to the user.

Figure 1 depicts an environment suitable for practicing the illustrative embodiment of the present invention. An electronic device 2 includes a modeling and execution environment 4. The electronic device 2 may be a server, client, workstation, laptop, PDA or other electronic device equipped with a processor. The modeling and execution environment, such as Simulink™ from The Mathworks, Inc. of Natick, Massachusetts, includes at least one graphical model 6 of a software diagram, such as a model of a dynamic system being modeled with a block diagram or a data flow diagram or other type of graphical program such as a Unified Modeling Language diagram. The modeling and execution environment 4 also includes an automatic code generator 7 used to programmatically generate source code referencing data of the graphical model 6. The graphical model 6 may graphically depict time-dependent mathematical relationships among the system's inputs, states and outputs. The graphical model 6 is an executable specification that can be translated to target ready code for a particular platform. Platforms may include CPUs, real-time operating systems, custom Application Specific Integrated Circuits (ASICs), and hardware Field Programmable Gate Arrays (FPGAs). The data referred to may encompass many different types of data. For example, in a block diagram, the source code may reference signal objects and the signal line values from signal lines connecting blocks in a block diagram. Alternatively, the source code may reference block objects and block values and functionality of a block from the block diagram.

The electronic device 2 is interfaced with a display 10. The display 10 includes a user interface 12 generated by the modeling and execution environment 4. The user interface 12, which is accessible by a user 30 through an input device such as a mouse, includes user-selectable parameter settings 14, 16 and 18. The user selectable parameter settings 14, 16 and 18 are parameters for use in the creation of a custom storage class 8 and are discussed in more detail below. The user-selectable parameter settings 14, 16 and 18 may be represented on the user interface 12 in a number of ways including through the use of conventional user interface controls such as radio buttons, check boxes or text boxes able to accept textual input specifying a parameter value. The user-selectable parameters are utilized by the automatic code generator 7 to create a custom class. The custom class is then utilized by the automatic code generator the create source code referencing data for the graphical model 6. The user interface 12 also includes a code view area 20 where the automatic code generator 7 may display salient aspect of the source code created from the use of the custom class.

Those skilled in the art will recognize that the environment depicted in **Figure 1** is just one of many environments falling within the scope of the present invention. For example, the various components depicted in **Figure 1** may be dispersed over a distributed network. Thus the user 30 and the display 10 may be located remotely from a server holding the modeling and execution environment 4. Similarly the graphical model 6 and the created custom storage class 8 may be located remotely from the modeling and execution environment 4 over a network. It will be appreciated that many other configurations are also possible within the scope of the present invention.

The graphical model 6 may be a block diagram model. A block diagram model includes a set of symbols, called blocks, interconnected by signal lines that carry signals. Blocks are functional entities that operate on signal values contained in the signal lines. Each block can have zero or more input signal lines and zero or more output signal lines. Blocks may also have states. A state is a variable that determines a block's output and whose current value is a function of the previous values of the block's states and/or inputs. In a block diagram model, the

signals carried on the signal lines are the streams of values that appear at the output ports of blocks. The signal can have a wide range of attributes, such as name, data type (e.g., 8-bit, 16-bit or 32-bit integer), numeric type (e.g., real or complex), and dimensionality (e.g., one-dimension array, two-dimension array or multi-dimensional array).

Each item of data in the graphical model 6 is defined to have a data storage class. A data storage class contains the instructions that define how the automatic code generator 7 is to produce code when a reference to an item of data belonging to that data storage class is encountered in the model. Data storage classes may be either pre-defined or custom. Pre-defined data storage classes represent those data storage classes inherent within a modeling system. Pre-defined data storage classes are not capable of modification by a user and represent an instruction or set of instructions for each type of reference to the data. It should be appreciated that although the examples contained herein are made with reference to block diagram models, the illustrative embodiment of the present invention may also be applied to other types of diagrams besides block diagrams.

Custom data storage classes are not inherent to the modeling system and are generated by an interpreted programming language process. Using the interpreted programming language process, the user specifies a custom data storage class by specifying a set of instructions for each type of reference to the data. This set defines the data storage class. Specifically, the custom data storage class is characterized by the set of instructions defined by the user via parameter settings that detail how to generate software source code for each type of reference to data that is of that class. The parameter settings for the custom class are chosen by the user via the user interface 12. The user may then specify that custom data storage class as the item's storage class. Because the user specifies and defines these instructions, the possible variations in the software source code to be generated are extensive. The instructions corresponding to a given custom data storage class may be a function of the set of items that are defined to be of that class, so that the mechanism is self-referential.

Once the graphical model 6 is specified, an automatic code generator 7 examines each data item specified in the graphical model 6 and determines the storage class of the data item. If the data storage class is predefined, the automatic code generator 7 applies a fixed set of instructions to generate code for each type of reference to the item. If the data storage class is determined to be a custom storage class 8, the automatic code generator 7 accesses an external set of user-defined instructions associated with the custom data storage class through an application programming interface (API) to generate code for each type of reference to the item.

An implementation of the code generation process can be seen in Real-Time Workshop™ from The MathWorks, Inc. of Natick, Massachusetts. Real-Time Workshop™ is a set of tools that generate code from Simulink™ models for targeted systems. Simulink™ translates the graphical model 6 into a Real-Time Workshop ".rtw" file. Simulink™ is a software package for modeling, simulating, and analyzing dynamic systems. Simulink™ supports linear and nonlinear systems, modeled in continuous time, sampled time, or a hybrid of the two. Systems can also be multi-rate, i.e., have different parts that are sampled or updated at different rates. Real-Time Workshop takes the .rtw file and pre-defined data storage classes supplied by The MathWorks, Inc., to produce C programming language code corresponding to the graphical model 6. During the code generation process, tokens in the pre-defined storage class files are replaced with strings defined in the .rtw file. Customized code generation occurs when Real-Time Workshop™ encounters a data record in the .rtw file that instructs it to utilize a custom data storage class instead of the predefined data storage class to generate code corresponding to a reference to data. More specifically, when an item of data is defined to have a custom data storage class, instructions associated with that data storage class are used to generate code corresponding to the data, at each point in the code generation process where the data is referenced.

As noted above, the illustrative embodiment of the present invention enables a user to customize the way source code is generated for a model by adjusting parameter settings through a user interface. **Figure 2** depicts the sequence of steps

followed by the illustrative embodiment of the present invention to create and display source code referencing model data. The sequence begins when the user interface 12 is displayed containing multiple parameter settings 14, 16 and 18 (step 40). The user 40 selects parameters for the custom storage class (step 42). The custom storage class is created by utilizing the selected parameter settings 14, 16 and 18 (step 44). The automatic code generator 7 creates source code referencing the data of the graphical model 6 using the settings embodied in the custom storage class (step 46). The created source code is then displayed in the user interface 12 to the user 30 (step 48). The displayed source code may include tokens or regular expressions representative of actual code segments.

The initial generation of code responsive to the parameter settings may not always result in a desired outcome for the user. The illustrative embodiment of the present invention allows the process to iterate with a real time display of adjusted code reflective of each code recalculation. **Figure 3** is a flowchart of the sequence of steps followed by the illustrative embodiment of the present invention to adjust and re-display source code generated following a change in the selected parameters for a custom class. The sequence begins when the user selects parameters in the user interface 12 and the custom storage class is created (step 60). The custom storage class is then used to create source code referencing model data, and the source code is displayed to the user 30 in the code view 20 portion of the user interface 12(step 62). Subsequently, the user 30 may change parameter settings and the process creates new source code (step 64). The new source code based on the adjusted parameter settings is then displayed to the user 30 (step 66).

Figure 4 depicts a view 80 of the user interface 12 of the illustrative embodiment of the present invention. The view 80 includes a listing of different types of user selectable custom storage classes 82 (BitField, Const, ConstVolatile, Define, ExportToFile, ImportFromFile, Struct). Each type of custom storage class has multiple user-selectable parameter settings. For example the view 80 displays the general parameter settings for the Define storage class including a visibility parameter setting 84, a memory access setting 86, an initialization parameter setting 88, a constant parameter setting 90, a volatile parameter setting 92 and a qualifier

parameter setting 94. Those skilled in the art will notice that the parameter settings displayed on the user interface include pull down menus 84, 86 and 88, check boxes 90 and 92 and text boxes 94 through which a user may specify parameter settings. The view 80 also includes a validation result area 102 and a code preview area 100. The validation result displays the result of a check to make sure selected parameter setting may be implemented without by the automatic code generator 7. For example, some parameter settings may conflict and prevent code generation. The code preview area is the area of the user interface where the automatic code generator displays generated source code and symbolic representations of source code.

Those skilled in the art will recognize that other additional parameter settings may also be adjusted through the user interface 12. Header file information may be specified, as may certain details of Simulink™ usage (e.g. the user may specify whether parameter or signal objects will utilize the custom storage class). Various other settings specific to particular custom storage classes may also be specified (e.g. struct data settings for the Struct custom storage class). Certain non-portable directives to a compiler referred to as "pragmas", such as *near* and *far* qualifiers may also be specified via the user interface 12. It will be appreciated that additional custom storage classes may also be created and displayed via the user interface 12 in order for a user to adjust parameter settings for the code generation process.

Since certain changes may be made without departing from the scope of the present invention, it is intended that all matter contained in the above description or shown in the accompanying drawings be interpreted as illustrative and not in a literal sense. Practitioners of the art will realize that the system configurations depicted and described herein are examples of multiple possible system configurations that fall within the scope of the current invention. Likewise, the sequences of steps discussed herein are examples and not the exclusive sequence of steps possible within the scope of the present invention.